Group Members:

Ismael Barajas - ismaelbarajas30@csu.fullerton.edu

Patrick Mahoney - patrick.mahoney@csu.fullerton.edu

Zach Sarvas - zsarvas@csu.fullerton.edu

About:

This project creates a Twitter-like microblogging service where users can post messages, repost messages, view global messages, see a timeline of messages from other users, narrow timelines to consist of only users they are following, follow or unfollow users, update user profile info, see who you are following and who is following you, like messages, create polls, and view/respond to polls. This service is implemented using 5 RESTful back-end services which make use of SQL databases, Redis, DynamoDB, http basic authentication, and a service registry.

How To Run:

- Please make sure you are running the latest version of Redis/Hiredis, some features used are not compatible with older versions of Redis.
- Set up DynamoDB locally
- Navigate to your DynamoDB server folder and run this command to start DynamoDB server:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

- Run ./bin/dynamoDB_init.py from the command line to create tables for the Dynamo DataBase
- Replace your haproxy.cfg file with the provided file found in the “etc” folder called “haproxy.cfg”.

- Run ./bin/init.sh from the command line to create SQL databases.
- Run ./bin/foremanStart.sh from the command line to start foreman.
- Run sudo systemctl restart haproxy.

After following the above steps, the five microservices are now running as users.localhost, timelines.localhost, likes.localhost, and polls.localhost respectively. Registry is the only microservice not connected to the haproxy load balancer: http://localhost:5000.

API Documentation:

- ## Like Microservices:
    - **Connection:** close
    - **Server:** gunicorn/20.0.4
    - **content-type:** application/json; charset=utf-8
  - ### like(post_id, username):
    - **Description:** Takes **2** parameters **post_id** and **username**, where **username** is the user “liking” a post, and **post_id** is the post the user is “liking”. POSTs a “like” under the account **username** for the post with the specified **post_id**.
    - **Endpoint**: /like/
    - **Method**: POST
    - **Parameters**: **username** (text), **post_id** (number)
    - **Sample Request:**

```
http POST likes.localhost/like username='ProfAvery' post_id=1
```

    - **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "ProfAvery has liked /posts/1"
}
```

    - **Error Response:**

```
HTTP/1.1 403 Forbidden
{
        "message": "ProfAvery has already like /posts/1 ",
        "status": "403 Forbidden"
}
```

  - ### unlike(post_id, username):

- **Description:** Takes **2** parameters **post_id** and **username**, where **username** is the user “unliking” a post, and **post_id** is the post the user is “unliking”. DELETEs a “like” from the post with the specified **post_id** using the account **username**.
- **Endpoint**: /unlike/
- **Method**: DELETE
- **Parameters**: **username** (text), **post_id** (number)
- **Sample Success Request:**

```
http DELETE likes.localhost/unlike username='ProfAvery' post_id=1
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "ProfAvery has unliked /posts/1"
}
```

- **Sample Error Request:**

```
http DELETE likes.localhost/unlike username='KevinAWortman'
post_id=5
```

  - **Error Response:**

```
HTTP/1.1 403 Forbidden
{
        "message": "KevinAWortman has not liked /posts/5",
        "status": "403 Forbidden"
}
```

- **post_likes(post_id):**
  - **Description:** Takes **1** parameter **post_id** which is the ID of the post. GETs the total likes for the post with the specified **post_id**.
  - **Endpoint**: /like/<post_id>
  - **Method**: GET

- **Parameters**: **post_id** (number)
- **Sample Request:**

```
http GET likes.localhost/like/1
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "likes": 1,
        "post_id": "/posts/1"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "/posts/8 does not exist.",
        "status": "404 Not Found"
}
```

- **user_likes(username):**
  - **Description:** Takes **1** parameter **username**, and checks if the user exists. GETs the liked posts .
  - **Endpoint**: /like/posts/<username>
  - **Method**: GET
  - **Parameters**: **username** (text)
  - **Sample Request:**

```
http GET likes.localhost/like/posts/ProfAvery
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
```

```
        "liked_posts":
        [
                "/posts/1"
        ],
        "username": "ProfAvery"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAver does not exist.",
        "status": "404 Not Found"
}
```

- **popular_posts( ):**
  - **Description:** Takes **no** parameters. GETs the top 5 posts (out of all posts) with the greatest amount of likes.
  - **Endpoint**: /like/popular
  - **Method**: GET
  - **Parameters**: NA
  - **Sample Request:**

```
http GET likes.localhost/like/popular
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "popular_posts": [
                {
                        "/posts/1": 2
                },
```

```
                {
                        "/posts/2": 1
                }
        ]
}
```

- **Error Response: NA**

• Poll Microservices:

○ create_poll(username, question, options):

- **Description:** Takes **6** parameters **username, question, option_1, option_2, option_3,** and **option_4**, where **username** is the user posting a poll, **question** is the question other users will answer, and the **option variables** are the choices other users can select. POSTs a poll with the given **question**, **options**, and assigned **poll_id** if the **username** does not already exist in **users_voted**.
- **Endpoint**: /poll/
- **Method**: POST
- **Parameters**: **username** (text)**, question** (text)**, options** (multiple/array)
- **Sample Request:**

```
http POST polls.localhost/poll username='ProfAvery' question='test question' options:='["is that a plane? Nope just an option", "There goes another option", "Another option", "this is an option"]'
```

- **Success Response:**

```
HTTP/1.0 200 OK

{
  "message": "ProfAvery has created a Poll",
  "poll": {
    "details": {
      "options": [
        {
          "option_1": "is that a plane? Nope just an option",
          "votes": 0
        },
        {
          "option_2": "There goes another option",
          "votes": 0
```

```
            },
            {
               "option_3": "Another option",
               "votes": 0
            },
            {
               "option_4": "this is an option",
               "votes": 0
            }
         ],
         "question": "test question",
         "users_voted": {}
      },
      "poll_id": "1",
      "username": "ProfAvery"
   }
}
```


- **Error Response:**


```
HTTP/1.0 400 Bad Request

{
   "message": "Please provide at least 2 to 4 poll options.",
   "status": "400 Bad Request"
}
```


- **results_poll (username, poll_id):**
  - **Description:** Takes 2 parameters **username** and **poll_id**, where **username** is the name of the **user**, and **poll_id** is the poll ID of that user's **poll**. GETs the poll by the **username** and **poll_id**.
  - **Endpoint**: /poll/results/<username>/<poll_id>
  - **Method**: GET
  - **Parameters**: **username** (text), **poll_id (**number**)**
  - **Sample Request:**

```
http GET polls.localhost/poll/results/ProfAvery/1
```

  - **Success Response:**


```
HTTP/1.0 200 OK
```

```
{
  "details": {
    "options": [
      {
        "option_1": "is that a plane? Nope just an option",
        "votes": "0"
      },
      {
        "option_2": "There goes another option",
        "votes": "0"
      },
      {
        "option_3": "Another option",
        "votes": "0"
      },
      {
        "option_4": "this is an option",
        "votes": "0"
      }
    ],
    "question": "test question",
    "users_voted": {}
  },
  "poll_id": "1",
  "username": "ProfAvery"
}
```


- **Error Response:**

```
HTTP/1.0 404 Not Found
```

```
{
   "message": "The poll you are looking for does not exist.",
   "status": "404 Not Found"
}
```

- **update_poll(poll_id, owner_username, voter_username, vote):**
  - **Description:** Takes 4 parameters **poll_id**, **owner_username, voter_username,** and **vote**. Where **poll_id** is the ID of the poll being voted on, **owner_username** is the owner of the poll, **voter_username** is the user voting in the poll, and **vote** is the number option **voter_username** is choosing (option: 1-4). PUTs the **vote** . # include what is being stored for keeping track of vote count/user
  - **Endpoint**: /poll/vote/
  - **Method**: PUT
  - **Parameters**: **poll_id** (number)**, owner_username** (text)**, voter_username** (text)**, vote** (number)
  - **Sample Request:**

```
http PUT polls.localhost/poll/vote poll_id='1'
owner_username='ProfAvery' voter_username='TestUser' vote=3
```

  - **Success Response:**

```
HTTP/1.0 200 OK
{
   "message": "Your vote for option 3 has been taken."
}
```

  - **Error Response:**

```
HTTP/1.0 403 Forbidden

{
   "message": "TestUser has already voted in ProfAvery's poll.",
   "status": "403 Forbidden"
}
```

- **delete_poll(username, poll_id):**
  - **Description:** Takes 2 parameters **username** and **poll_id**, where **username** is the user who owns the poll, and **poll_id** is the ID of the poll being deleted. DELETEs the poll with the given **username** and **poll_id**.
  - **Endpoint**: /poll/delete
  - **Method**: DELETE
  - **Parameters**: **username** (text), **poll_id** (number)
  - **Sample Request:**

```
http DELETE polls.localhost/poll/delete poll_id='1'
username='ProfAvery'
```

  - **Success Response:**

```
HTTP/1.0 200 OK
{
   "message": "ProfAvery has successfully deleted poll_id:1."
}
```

  - **Error Response:**

```
HTTP/1.0 404 Not Found

{
   "message": "Poll you are trying to delete does not exist.",
   "status": "404 Not Found"
}
```

• Service Registry Microservices (localhost:5000):

- **register(service, url):**
  - **Description:** Takes 2 parameters **service** and **url**, where **service** is the name of the service (e.g. 'users', 'posts', or 'likes'), and **url** is the url of the service being registered. POSTs the **url** for the **service**.
  - **Endpoint**: /register
  - **Method**: POST

- **Parameters**: **service** (text), **url** (text)
- **Sample Request:**

```
http POST localhost:5000/register service='timelines'
url='http://timelines.localhost '
```

- **Success Response:**

```
HTTP/1.0 200 OK
{
   "likes": {
      "http://127.0.1.1:5300": true
   },
   "polls": {
      "http://127.0.1.1:5400": true
   },
   "timelines": {
      "http://127.0.1.1:5200": true,
      "http://127.0.1.1:5201": true,
      "http://127.0.1.1:5202": true,
      "http://timelines.localhost": true
   },
   "users": {
      "http://127.0.1.1:5100": true
   }
}
```

- **Error Response:**

```
HTTP/1.0 422 Unprocessable Entity

{
   "message": "timelinesa does not exist.",
   "status": "422 Unprocessable Entity"
}
```

- **avaliable_services(service):**
  - **Description:** Takes 1 parameter **service**, which is the name of the service. GETs the availability of the service in the database.
  - **Endpoint**: /available/<service>
  - **Method**: GET
  - **Parameters**: **service** (text)
  - **Sample Request:**

```
http GET localhost:5000/available/timelines
```

  - **Success Response:**

```
HTTP/1.0 200 OK
{
   "timelines": {
      "http://127.0.1.1:5200": true,
      "http://127.0.1.1:5201": true,
      "http://127.0.1.1:5202": true
   }
}
```

  - **Error Response:**

```
HTTP/1.0 404 Not Found
{
   "message": "timelineasd does not exist.",
   "status": "404 Not Found"
}
```

- **health_check():**
  - On startup, registry starts a daemon thread targeting the health_check() function which runs health checks on the self registering services every 20 seconds. When a request fails the service url that failed gets removed from the available services dictionary.

```
registry = {
    "users": {},
    "timelines": {},
    "likes": {},
    "polls": {}
}

# Runs health checks on services
def health_check():
    _lock = threading.Lock()
    while 1:
        for api in registry:
            for url in copy.deepcopy(registry[api]):
                try:
                    r = requests.get(f"{url}/health-check")
                    r.raise_for_status()
                except requests.HTTPError:
                    with _lock:
                        del registry[api][url]
        time.sleep(20)

# Starts daemon thread that runs forever, that checks health of all services
@hug.startup()
def run_heath_check(api):
    threading.Thread(target=health_check, daemon=True).start()
```